Name : VEDASHREE BHALERAO

Class : LY – AIA – 1

Batch : B

Roll No : 2213191

Assignment No. 5

Aim: Write a program to demonstrate the change in accuracy/loss/convergence time with change in optimizers like stochastic gradient descent, adam, adagrad, RMSprop and Nadam for any suitable application **Objectives:**

1. To learn optimization algorithms 2. To

learn and understand hyperparameters

Theory:

SGD, Adam, RMSprop, Nadam

The word '*stochastic*'means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets gets big.

Adam is a replacement optimization algorithm for stochastic gradient descent for training **deep learning** models. **Adam** combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated. The following equations show how the gradients are calculated for the RMSprop and gradient descent with momentum. The value of momentum is denoted by beta and is usually set to 0.9.

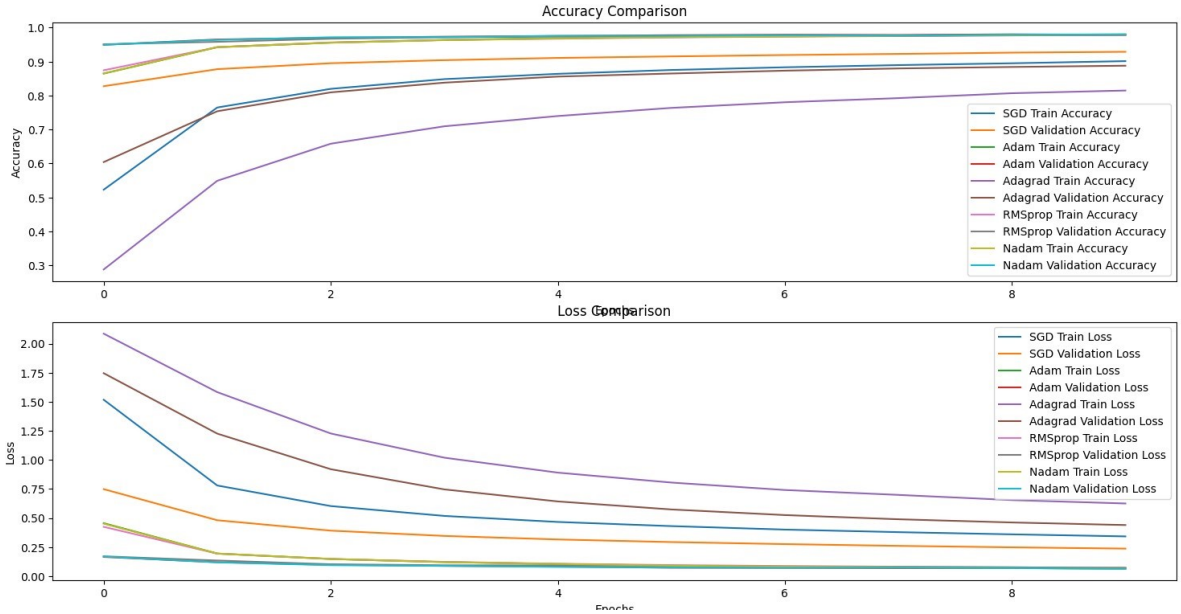
Nadam combines NAG and Adam. Nadam is employed for noisy gradients or for gradients with high curvatures. The learning process is accelerated by summing up the exponential decay of the moving averages for the previous and current gradient

Code:

```
import tensorflow as tf from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout from
tensorflow.keras.optimizers import SGD, Adam, Adagrad, RMSprop, Nadam from
tensorflow.keras.datasets import mnist from tensorflow.keras.utils import
to_categorical import matplotlib.pyplot as plt import time

# Load and preprocess MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.astype("float32") / 255.0 x_test =
x_test.astype("float32") / 255.0 y_train =
to_categorical(y_train, 10) y_test =
to_categorical(y_test, 10)

# Define the neural network model
def create_model(optimizer):
model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(128, activation='relu'),
        Dropout(0.2),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(10, activation='softmax')
    ])     model.compile(optimizer=optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])     return model

# Optimizers to be tested
optimizers = {     'SGD':
SGD(),
    'Adam': Adam(),
    'Adagrad': Adagrad(),
    'RMSprop': RMSprop(),
    'Nadam': Nadam()
}

# Store results results
= {}

# Train the model with different optimizers and record time, accuracy, and
loss for name, optimizer in optimizers.items():
    print(f"Training with {name} optimizer...")
start_time = time.time()
         model = create_model(optimizer)     history = model.fit(x_train,
y_train, validation_data=(x_test, y_test), epochs=10, batch_size=128,
verbose=0)
         end_time = time.time()
training_time = end_time - start_time
```

```
results[name] = {
        'history': history,
        'training_time': training_time
    }         print(f"{name} optimizer took {training_time:.2f} seconds to
converge.\n")
# Plot the results (Accuracy and Loss) fig, axes
= plt.subplots(2, 1, figsize=(10, 12))
# Plot Accuracy for name, result in
results.items():
    axes[0].plot(result['history'].history['accuracy'], label=f"{name} Train
Accuracy")
    axes[0].plot(result['history'].history['val_accuracy'], label=f"{name}
Validation Accuracy") axes[0].set_title('Accuracy Comparison')
axes[0].set_xlabel('Epochs') axes[0].set_ylabel('Accuracy')
axes[0].legend()

# Plot Loss for name, result in
results.items():
    axes[1].plot(result['history'].history['loss'], label=f"{name} Train Loss")
    axes[1].plot(result['history'].history['val_loss'], label=f"{name}
Validation Loss") axes[1].set_title('Loss Comparison')
axes[1].set_xlabel('Epochs') axes[1].set_ylabel('Loss')
axes[1].legend()

plt.tight_layout()
plt.show()

# Display training time for each optimizer
print("Training Time for each Optimizer:") for
name, result in results.items():
    print(f"{name}: {result['training_time']:.2f} seconds"
Result :
```

```
SGD optimizer took 13.52 seconds to converge.

Training with Adam optimizer...
Adam optimizer took 15.86 seconds to converge.

Training with Adagrad optimizer...
Adagrad optimizer took 14.59 seconds to converge.

Training with RMSprop optimizer...
RMSprop optimizer took 15.14 seconds to converge.

Training with Nadam optimizer...
Nadam optimizer took 16.63 seconds to converge.

Training Time for each Optimizer:
SGD: 13.52 seconds
Adam: 15.86 seconds
Adagrad: 14.59 seconds
RMSprop: 15.14 seconds
Nadam: 16.63 seconds
PS D:\2213133>
```



Conclusion:

Thus, we have understood the difference in performance of various optimisation algorithms